## Pogoji in zanke

Tema drugega predavanja so pogojni stavki in zanke. Kdor zna programirati, to že pozna. Razložil bom samo razliko med zapisom teh reči v jezikih, ki jih poznate, in zapisom v Pythonu, zraven pa še par Pythonovih posebnosti.

Tole je program, ki izračuna število členov Collatzovega zaporedja, ki se začne s podanim številom `n`. Kaj je Collatzovo zaporedje, ni pomembno.

```
n = 42
clenov = 1;
while (n != 1) {
    if (n % 2 == 0) {
        n /= 2;
    }
    else {
        n = 3 * n + 1;
    }
    clenov += 1;
}
```

To ni Python. Lahko bi bil C, C++, Javascript ... Približno katerikoli jezik, v katerem so vas učili programirati, če vas niso v Pythonu. Na tri stvari bomo pozorni.

- Program je sestavljen iz "blokov", ki jih označimo z zavitimi oklepaji.
- Pogoje v `while` in v `if` zapiramo v oklepaje.
- Na koncu vsakega stavka je podpičje.

Tule je ista reč v Pythonu.

```
n = 42
clenov = 1
while n != 1:
    if n % 2 == 0:
        n /= 2
    else:
        n = 3 * n + 1
    clenov += 1
```

**Prvo: Bloki in zamiki** Tudi programi v Pythonu imajo bločno strukturo, vendar brez zavitih oklepajev. Blok začnemo z dvopičjem. Potem pa veljajo zamiki. Medtem ko se jih moramo v Cju držati zato, da je program berljiv, se jih moramo v Pythonu držati zato, da je pravilen.

Spodnje ni OK, ker je `clenov += 1` nekje vmes - ne v `else`, ne zunaj:

```
clenov = 1
while n != 1:
    if n % 2 == 0:
```

```
        n /= 2
    else:
        n = 3 * n + 1
      clenov += 1
```

Spodnje ni OK, ker bi morala biti `if` in `else` na istem nivoju:

```
n = 42
clenov = 1
while n != 1:
    if n % 2 == 0:
        n /= 2
        else:
        n = 3 * n + 1
    clenov += 1
```

Spodnje ni OK, ker bi moral biti `n /= 2` zamaknjen:

```
n = 42
clenov = 1
while n != 1:
    if n % 2 == 0:
    n /= 2
    else:
        n = 3 * n + 1
    clenov += 1
return clenov
```

Vse to je očitno narobe. Vi samo pravilno zamikajte, pa bo vse v redu.

Praviloma zamikamo za štiri presledke. Če hočete, lahko tudi za dva ali samo enega, ampak to ni berljivo.

```
clenov = 1
while n != 1:
 if n % 2 == 0:
  n /= 2
 else:
  n = 3 * n + 1
 clenov += 1
```

Lahko tudi za petnajst, ampak to bi bilo videti tako:

```
clenov = 1
while n != 1:
               if n % 2 == 0:
                              n /= 2
               else:
                              n = 3 * n + 1
               clenov += 1
```

Presledki ali tabulatorji? Presledki. Tabulatorji so tolerirani. Mešanica pa je prepovedana - Python v tem primeru javi napako.

**Drugo: Oklepaji** Python poskuša biti zračen. Kot ste videli, pri `while` in `if` ne zapiramo pogojev v oklepaje, ker ... zakaj bi jih?

Mogoče se vam zdi, da je z njimi pregledneje. To se vam zdi zato, ker ste tako navajeni iz C-ja. V Pythonu pišite brez njih. Ker je tako zmenjeno. In hitro se vam bo zdelo v Pythonu pregledneje brez, v C-ju pa z. Dober programer prevzame navade jezika. V Pythonu pišemo oklepaje samo tam, kjer so potrebni, recimo zaradi prednosti operatorjev.

**Tretje: Konci vrstic** V Cju konec stavka označimo s podpičjem. V Pythonu konec stavka označimo tako, da gremo v novo vrstico. Zato ne smemo kar tako brez zveze lomiti vrstic.

V C-ju smemo napisati

```
while (n != 1) {
    if (n % 2 == 0) {
        n
        /= 2;
    }
```

V Pythonu pa ne. Tam, kjer gremo v novo vrstico, je tako, kot če bi v Cju naredili podpičje. No, obstajata dva načina, da prelomimo vrstico, ne da bi Python to dojel kot konec vrstice. Ampak to bomo videli sproti, ko bo potrebno.

Mogoče boste opazili, da lahko tudi v Pythonu uporabljate podpičja. Ostala so iz zgodovinskih razlogov. Noben resen programer v Pythonu jih ne piše. Tudi vi jih ne pišite.

## Pogojni stavki

Pogoje očitno pišemo z `if` in `else`. Tu pride do male zafrkancije, ko stvari nizamo. V Cju bi lahko pisali

```
if (nekaj) {
    ...
}
else if (nekaj_drugega) {
    ...
}
else if (nekaj_tretjega) {
    ...
}
else {
    ...
}
```

V Pythonu bi to izgledalo tako

```
if nekaj:
    ...
else:
    if nekaj_drugega:
        ...
    else:
        if nekaj_tretjega:
            ...
        else:
            ...
```

To ni kul, iz dveh razlogov. Prvi je, da lezemo vedno bolj na desno. Drugi je, da je v C-jevskem primeru program jasnejši: ima tri alternative (`nekaj`, `nekaj_drugega` in `nekaj_tretjega`) ter rezervni scenarij. Vsi so nekako na istem nivoju - zgodi se prvo, drugo ali tretje. V Pythonu pa je videti, kot da se situacija, kjer `nekaj` ni res, razdeli na dve podvarianti, in če `nekaj_drugega` ni res, imamo spet dve podvarianti.

Za take primere imamo v Pythonu poleg `if` in `else` še `elif`.

```
if nekaj:
    ...
elif nekaj_drugega:
    ...
elif nekaj_tretjega:
    ...
else:
    ...
```

## Zanke

Videli smo `while`. Ta deluje čisto tako kot `while` v drugih jezikih. Tudi `break` in `continue` imamo. Posebnost je, da ima `while` lahko še `else`. Ampak ta `else` je bolj zanimiv v kombinaciji s `for`, tako da ga bomo spoznali takrat, ko se bomo učili o `for`-u. Ta pa je precej drugačen od tistega, ki ste ga navajeni in bo vreden posebnega predavanja. Zanke `do - while` (ali `repeat - until`) pa v Pythonu ni. V resnici jo tudi v jezikih, ki jo imajo, zelo redko potrebujemo.

## Pogoji

`==`, `!=`, `<`, `>`, `<=` in `>=` delujejo tako, kot ste jih navajeni. Ena lepa stvar v Pythonu je, da jih lahko nizamo: `if a < b < c:` ali `if 10 <= a == b <= 20`. Slednje pomeni, da morata biti `a` in `b` enaka ter morata biti med `10` in `20`. Ali pa tole: `if x < 10 < y:`: `x` mora biti manjši od `10`, `y` pa večji. Možne so tudi poljubne bedaste kombinacije, `if 10 <= a >= b == 3 > x:`, ampak tega seveda ne počnemo. Celo če je slučajno smiselno in pravilno, ni berljivo.

S temi operatorji lahko primerjamo tudi nize in še vse žive druge reči. Če nize primerjamo po velikosti, jih primerja po abecedi. Vsaj dokler gre za črke angleške abecede.

Python se trudi biti berljiv in zračen, zato pogojev ne sestavljamo z `&&`, `||` in `!`, temveč `and`, `or` in `not`.

Konstanti `True` in `False` se pišeta z veliko začetnico. Kot sem ju pravkar napisal.

Zanimivost: razred `bool` je izpeljan iz razreda `int`. `True` je samo malo drugače izpisano število `1` in `False` je samo malo drugače izpisana `0`. `True + True` je 2. Ampak to je samo neuporabna zanimivost.

Pogoji se računajo samo, do koder je treba. `5 > 6 and 1 / 0 > 12` ne javi deljenja z 0 temveč `False`. Ker je že `5 > 6` neresnično in mu potem sledi `and`, Python ne vidi potrebe, da bi računal še naprej. (To je enako tudi v Cju in vseh drugih normalnih jezikih.) Podobno `5 < 6 or 1 / 0 > 12` ne vrne napake, ker je že `5 < 6` resnično in ne vidi potrebe, da bi gledal še desno od `or`-a.